EAST:

An Efficient and Accurate Scene Text Detector

Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang
Megvii Technology Inc., Beijing, China

Sungman, Cho.

text detection ?

Blurring, irregular text

Text detection - branch

- **Regression-based text detectors**

  DMPNet (CVPR, 2017) → TextBoxes (CVPR, 2018) → RSDD (CVPR, 2018)

- **Segmentation-based text detectors**

  SSTD (ICCV, 2017) → **EAST (CVPR, 2017)** → TextSnake (arXiv, 2018) → CRAFT (CVPR, 2019)

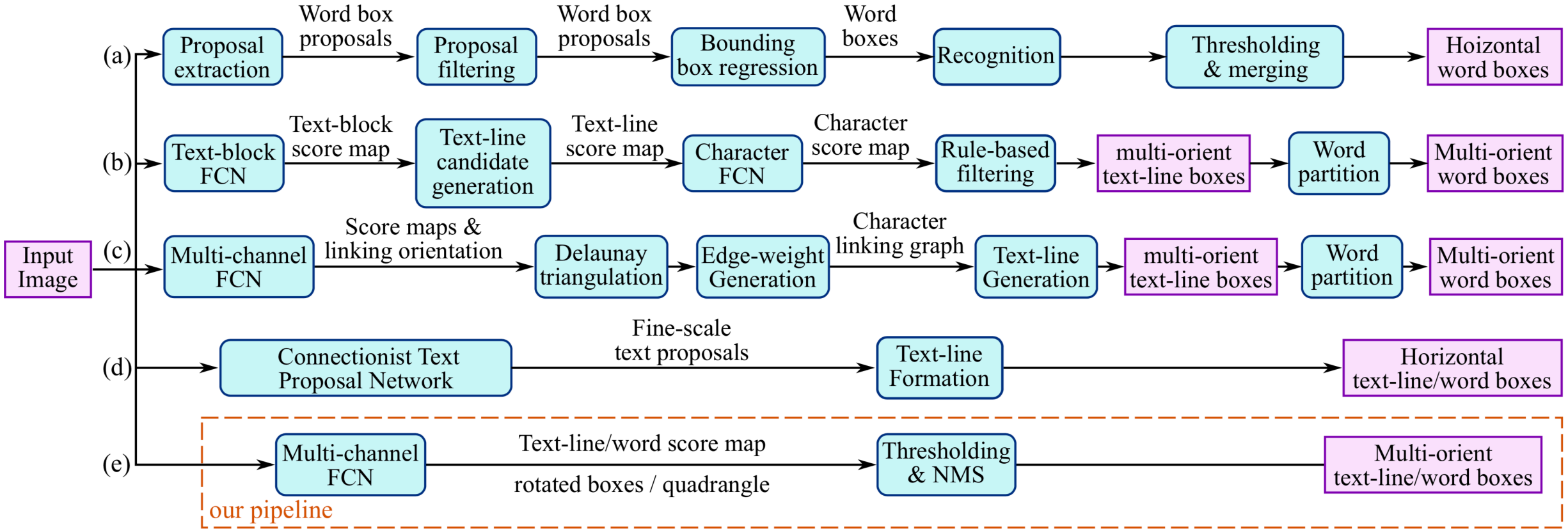
- **End-to-end text detectors**

  FOTS (CVPR, 2018) → EAA (CVPR, 2018) → Mask TextSpotter (arXiv, 2018)

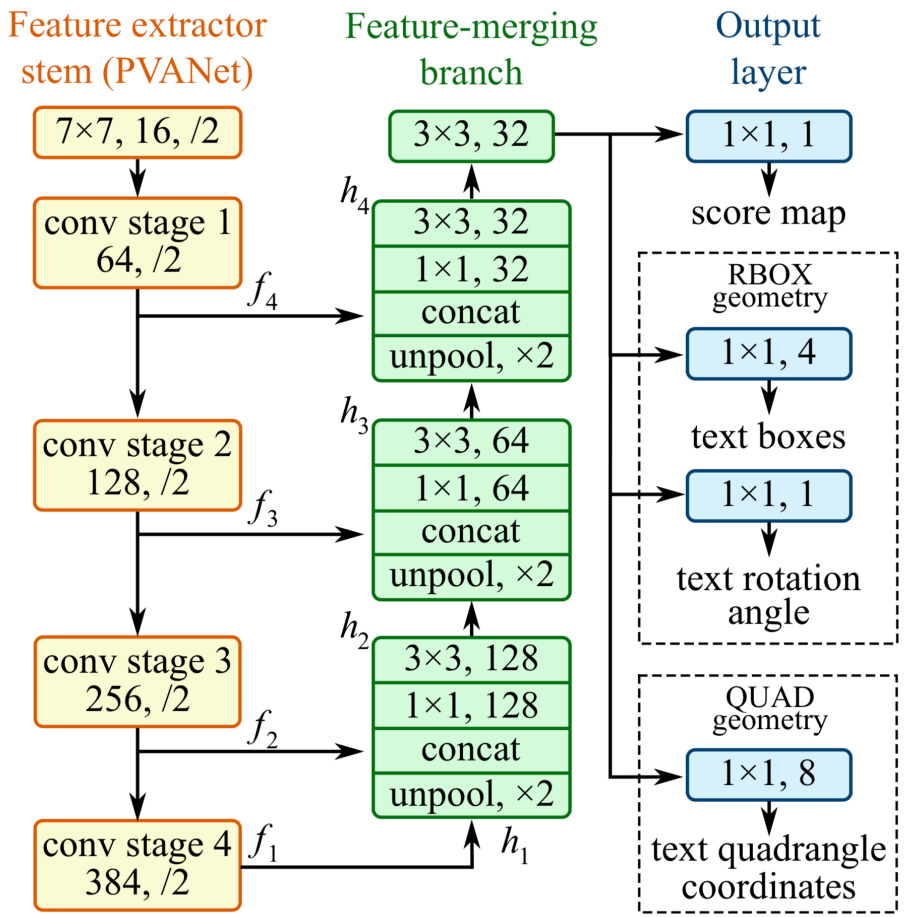
E A S T

Architecture.

EAST : simple architecture !

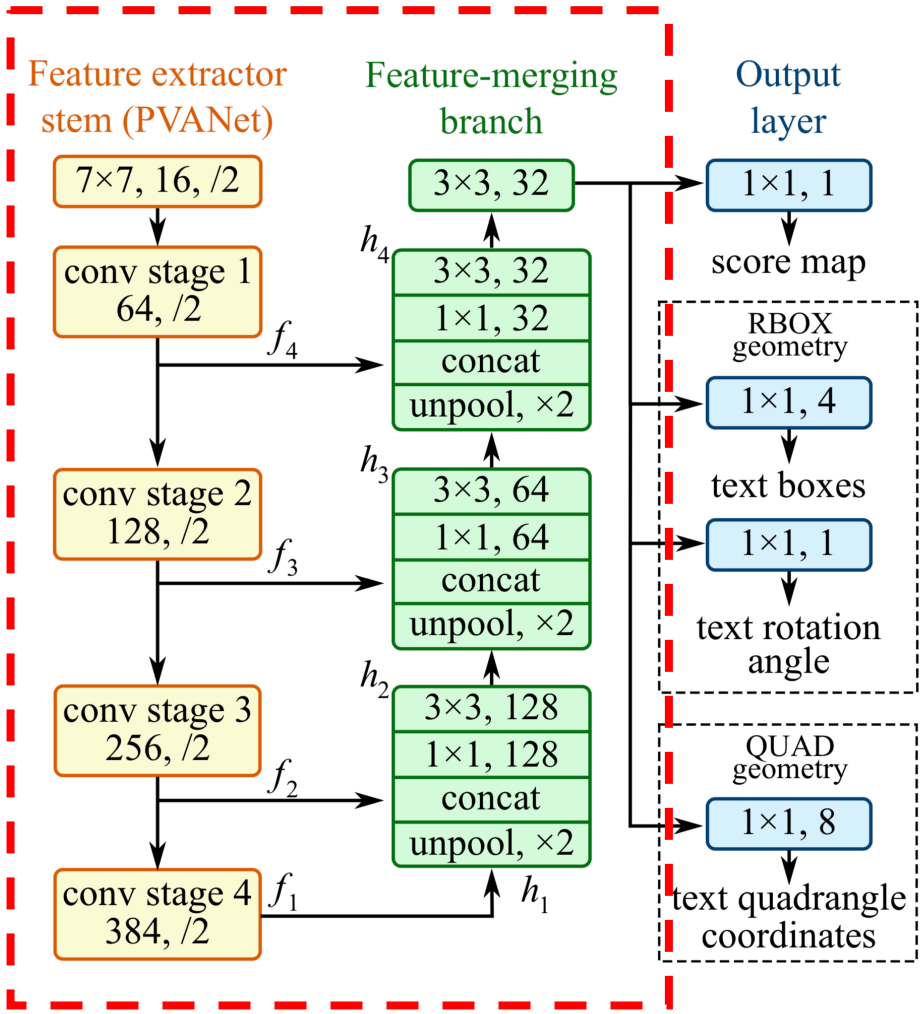


stem + branch + output



- FCN based text-detection.
- Eliminates intermediate steps such as candidate proposal.
- The post-processing steps only include thresholding and NMS.

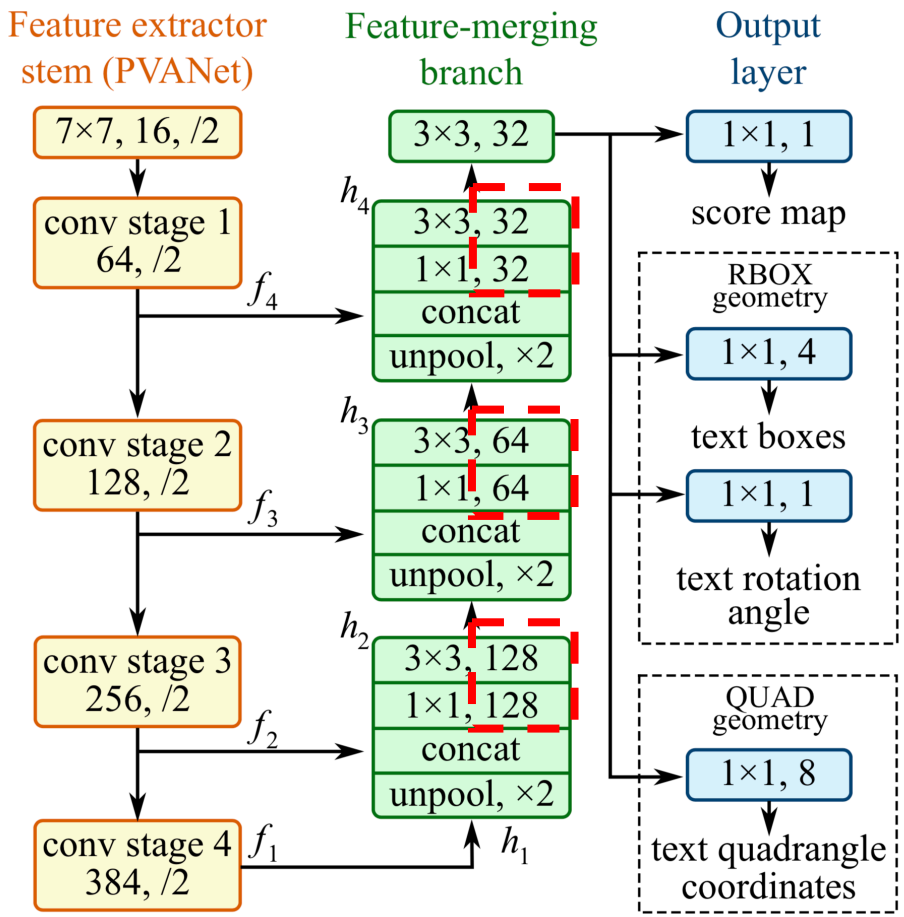
Like a U-NET



U-NET Architecture

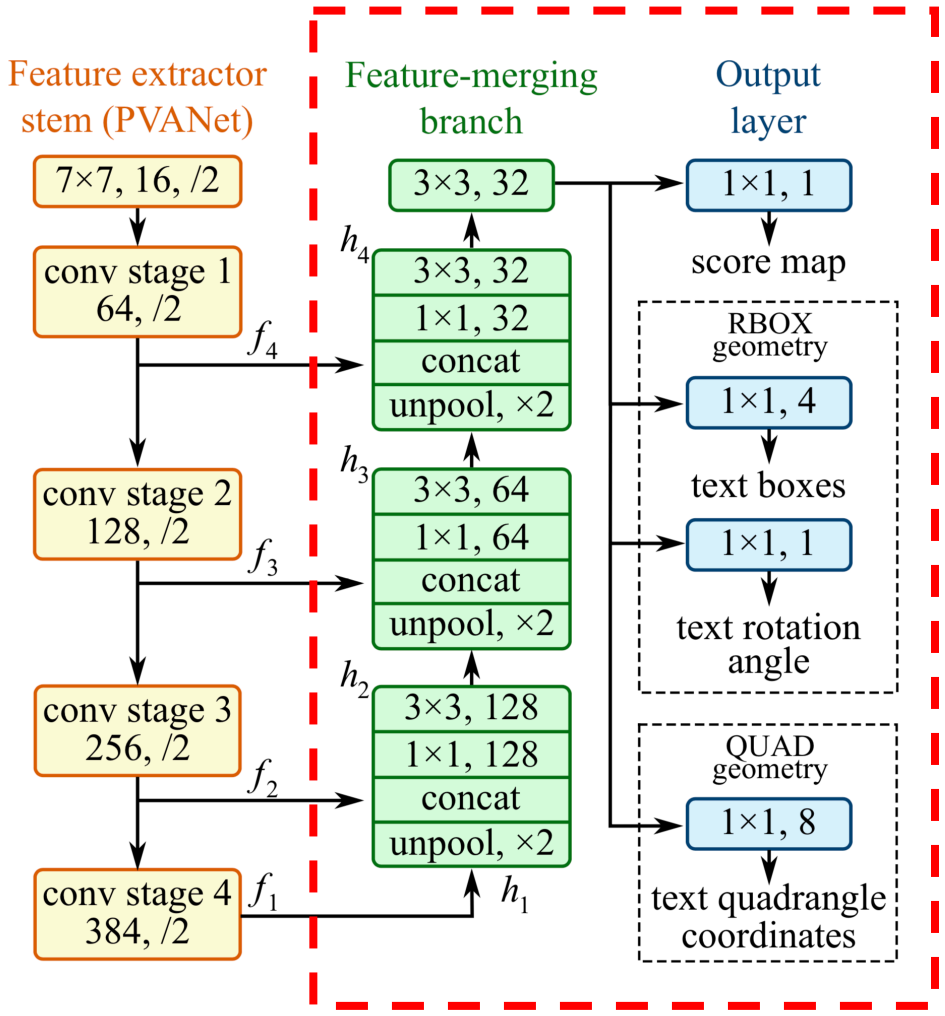
- FCN based text-detection.
- Eliminates intermediate steps such as candidate proposal.
- The post-processing steps only include thresholding and NMS.

Computation-efficient



- Keep the number of channels for convolutions in *branch* small, making the network computation-efficient

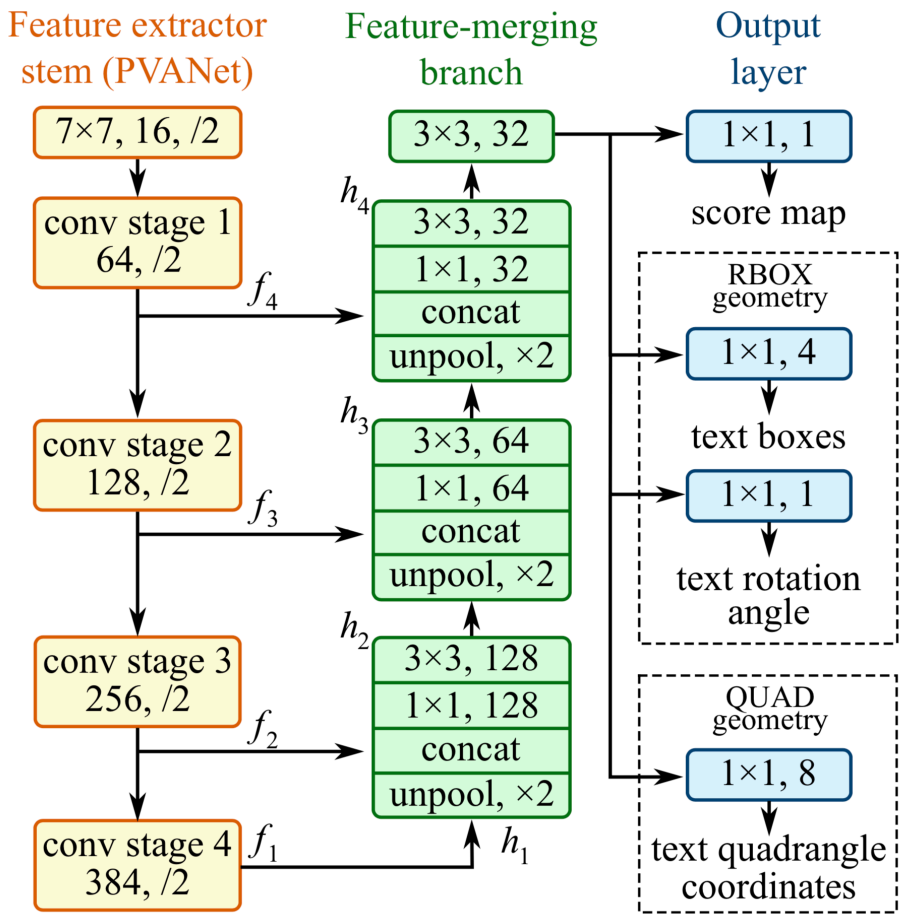
The final output layer



- The geometry output can be either one of RBOX or QUAD

Geometry	channels	description
AABB	4	$\mathbf{G} = \mathbf{R} = \{d_i \mid i \in \{1, 2, 3, 4\}\}$
RBOX	5	$\mathbf{G} = \{\mathbf{R}, \theta\}$
QUAD	8	$\mathbf{G} = \mathbf{Q} = \{(\Delta x_i, \Delta y_i) \mid i \in \{1, 2, 3, 4\}\}$

The final output layer

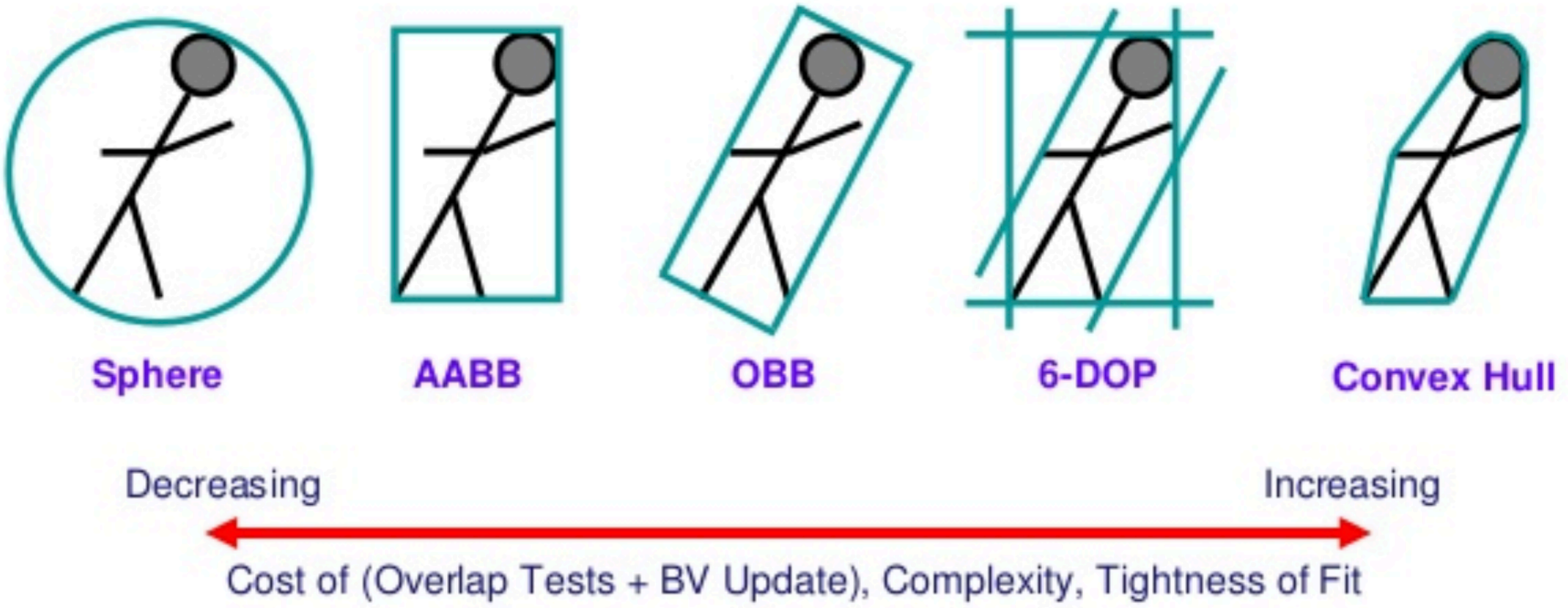


- The geometry output can be either one of RBOX or QUAD

Geometry	channels	description
AABB	4	$\mathbf{G} = \mathbf{R} = \{d_i \mid i \in \{1,2,3,4\}\}$
RBOX	5	$\mathbf{G} = \{\mathbf{R}, \theta\}$
QUAD	8	$\mathbf{G} = \mathbf{Q} = \{(\Delta x_i, \Delta y_i) \mid i \in \{1,2,3,4\}\}$

AABB? RBOX? QUAD?

Appendix : Type of bounding boxes



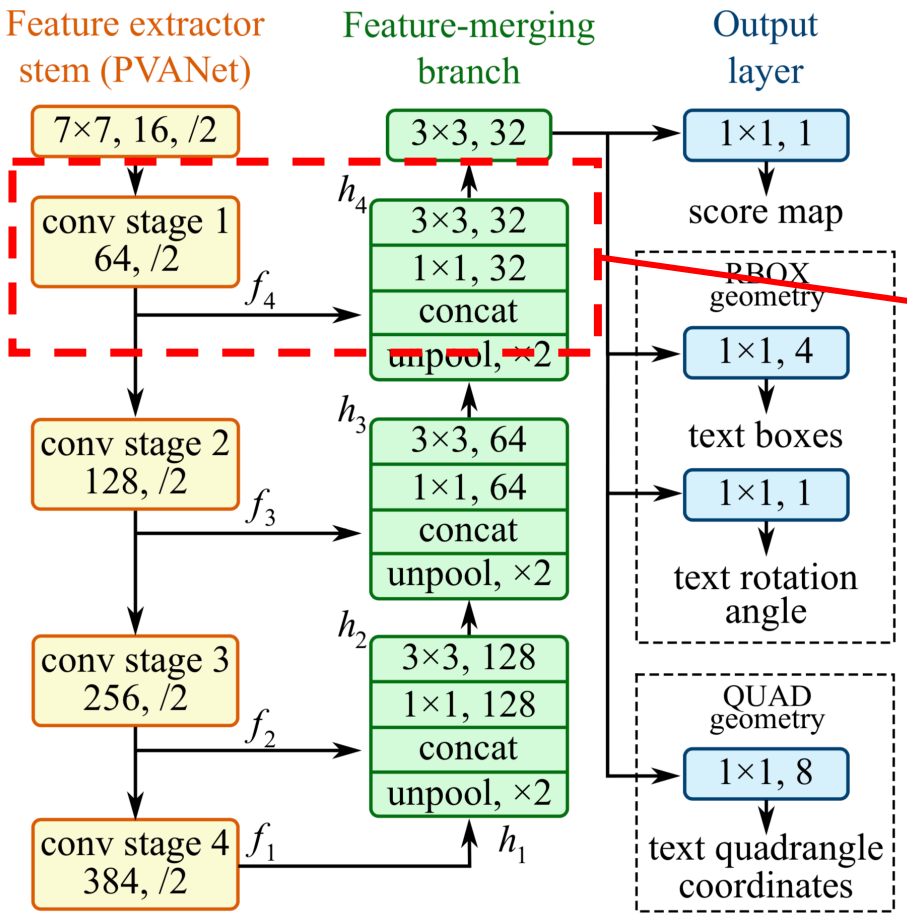
```python
class East(nn.Module):
    def __init__(self):
        super(East, self).__init__()
        self.resnet = resnet50(True)
        self.conv1 = nn.Conv2d(3072, 128, 1)
        self.bn1 = nn.BatchNorm2d(128)
        self.relu1 = nn.ReLU()

        self.conv2 = nn.Conv2d(128, 128, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.relu2 = nn.ReLU()

        self.conv3 = nn.Conv2d(640, 64, 1)
        self.bn3 = nn.BatchNorm2d(64)
        self.relu3 = nn.ReLU()

        self.conv4 = nn.Conv2d(64, 64, 3 ,padding=1)
        self.bn4 = nn.BatchNorm2d(64)
        self.relu4 = nn.ReLU()

        self.conv5 = nn.Conv2d(320, 64, 1)
        self.bn5 = nn.BatchNorm2d(64)
        self.relu5 = nn.ReLU()

        self.conv6 = nn.Conv2d(64, 32, 3, padding=1)
        self.bn6 = nn.BatchNorm2d(32)
        self.relu6 = nn.ReLU()

        self.conv7 = nn.Conv2d(32, 32, 3, padding=1)
        self.bn7 = nn.BatchNorm2d(32)
        self.relu7 = nn.ReLU()

        self.conv8 = nn.Conv2d(32, 1, 1)
        self.sigmoid1 = nn.Sigmoid()
        self.conv9 = nn.Conv2d(32, 4, 1)
        self.sigmoid2 = nn.Sigmoid()
        self.conv10 = nn.Conv2d(32, 1, 1)
        self.sigmoid3 = nn.Sigmoid()
        self.unpool1 = nn.Upsample(scale_factor=2, mode='bilinear')
        self.unpool2 = nn.Upsample(scale_factor=2, mode='bilinear')
        self.unpool3 = nn.Upsample(scale_factor=2, mode='bilinear')
```

```python
def forward(self,images):
    images = mean_image_subtraction(images)
    _, f = self.resnet(images)
    h = f[3]  # bs 2048 w/32 h/32
    g = (self.unpool1(h)) #bs 2048 w/16 h/16
    c = self.conv1(torch.cat((g, f[2]), 1))
    c = self.bn1(c)
    c = self.relu1(c)

    h = self.conv2(c)  # bs 128 w/16 h/16
    h = self.bn2(h)
    h = self.relu2(h)
    g = self.unpool2(h) # bs 128 w/8 h/8
    c = self.conv3(torch.cat((g, f[1]), 1))
    c = self.bn3(c)
    c = self.relu3(c)

    h = self.conv4(c)  # bs 64 w/8 h/8
    h = self.bn4(h)
    h = self.relu4(h)
    g = self.unpool3(h) # bs 64 w/4 h/4
    c = self.conv5(torch.cat((g, f[0]), 1))
    c = self.bn5(c)
    c = self.relu5(c)

    h = self.conv6(c) # bs 32 w/4 h/4
    h = self.bn6(h)
    h = self.relu6(h)
    g = self.conv7(h) # bs 32 w/4 h/4
    g = self.bn7(g)
    g = self.relu7(g)

    F_score = self.conv8(g) #  bs 1 w/4 h/4
    F_score = self.sigmoid1(F_score)
    geo_map = self.conv9(g)
    geo_map = self.sigmoid2(geo_map) * 512
    angle_map = self.conv10(g)
    angle_map = self.sigmoid3(angle_map)
    angle_map = (angle_map - 0.5) * math.pi / 2

    F_geometry = torch.cat((geo_map, angle_map), 1) # bs 5 w/4 w/4
    return F_score, F_geometry
```

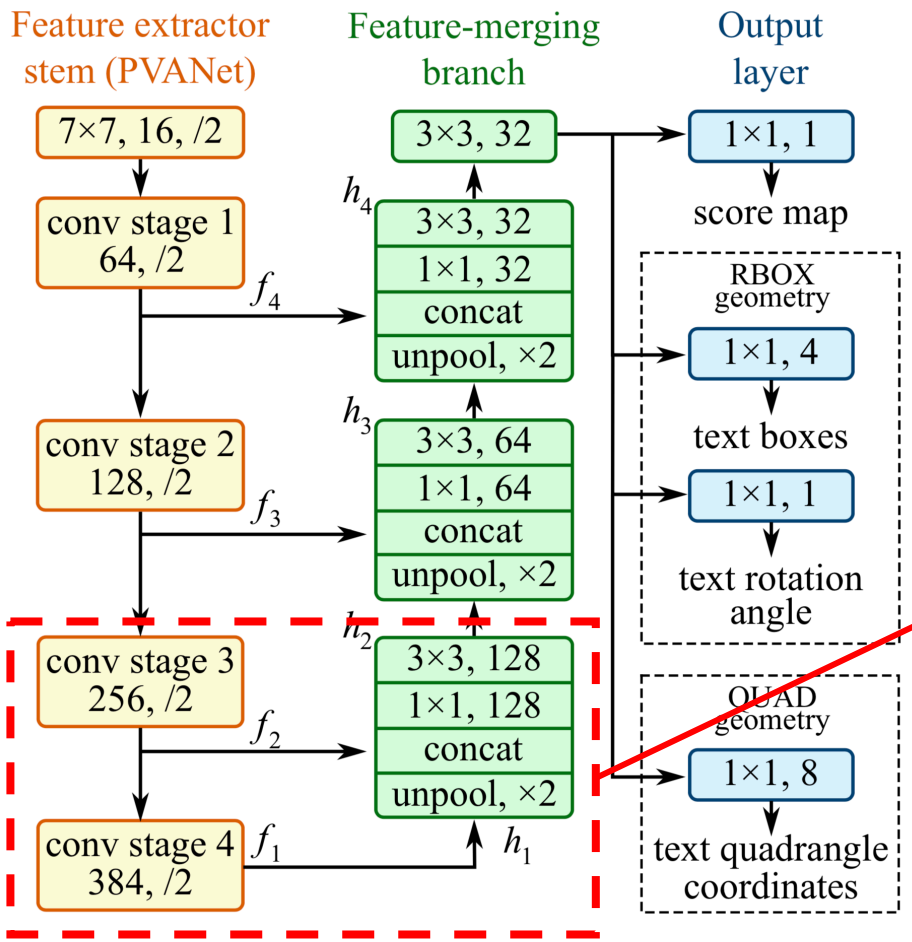
Why U-NET ?

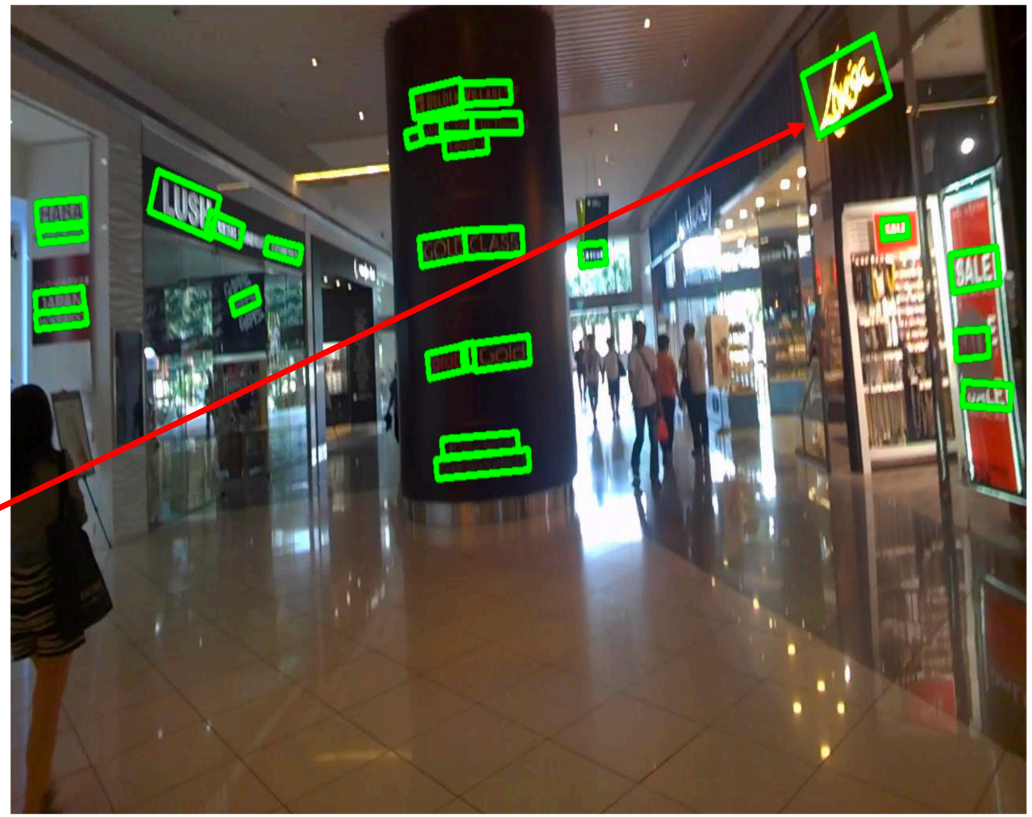
Sizes of word regions vary tremendously

Why U-NET ?


Feature extractor stem (PVANet)
Feature-merging branch
Output layer
7×7, 16, /2
conv stage 1 64, /2
conv stage 2 128, /2
conv stage 3 256, /2
conv stage 4 384, /2
f_4
f_3
f_2
f_1
h_4
h_3
h_2
h_1
3×3, 32
3×3, 32
1×1, 32
concat
unpool, ×2
3×3, 64
1×1, 64
concat
unpool, ×2
3×3, 128
1×1, 128
concat
unpool, ×2
1×1, 1
score map
RBOX geometry
1×1, 4
text boxes
1×1, 1
text rotation angle
QUAD geometry
1×1, 8
text quadrangle coordinates

Why U-NET ?


Feature extractor stem (PVANet)
Feature-merging branch
Output layer
7×7, 16, /2
conv stage 1 64, /2
conv stage 2 128, /2
conv stage 3 256, /2
conv stage 4 384, /2
f_4
f_3
f_2
f_1
h_4
h_3
h_2
h_1
3×3, 32
3×3, 32
1×1, 32
concat
unpool, ×2
3×3, 64
1×1, 64
concat
unpool, ×2
3×3, 128
1×1, 128
concat
unpool, ×2
1×1, 1
score map
RBOX geometry
1×1, 4
text boxes
1×1, 1
text rotation angle
QUAD geometry
1×1, 8
text quadrangle coordinates



Label Generation

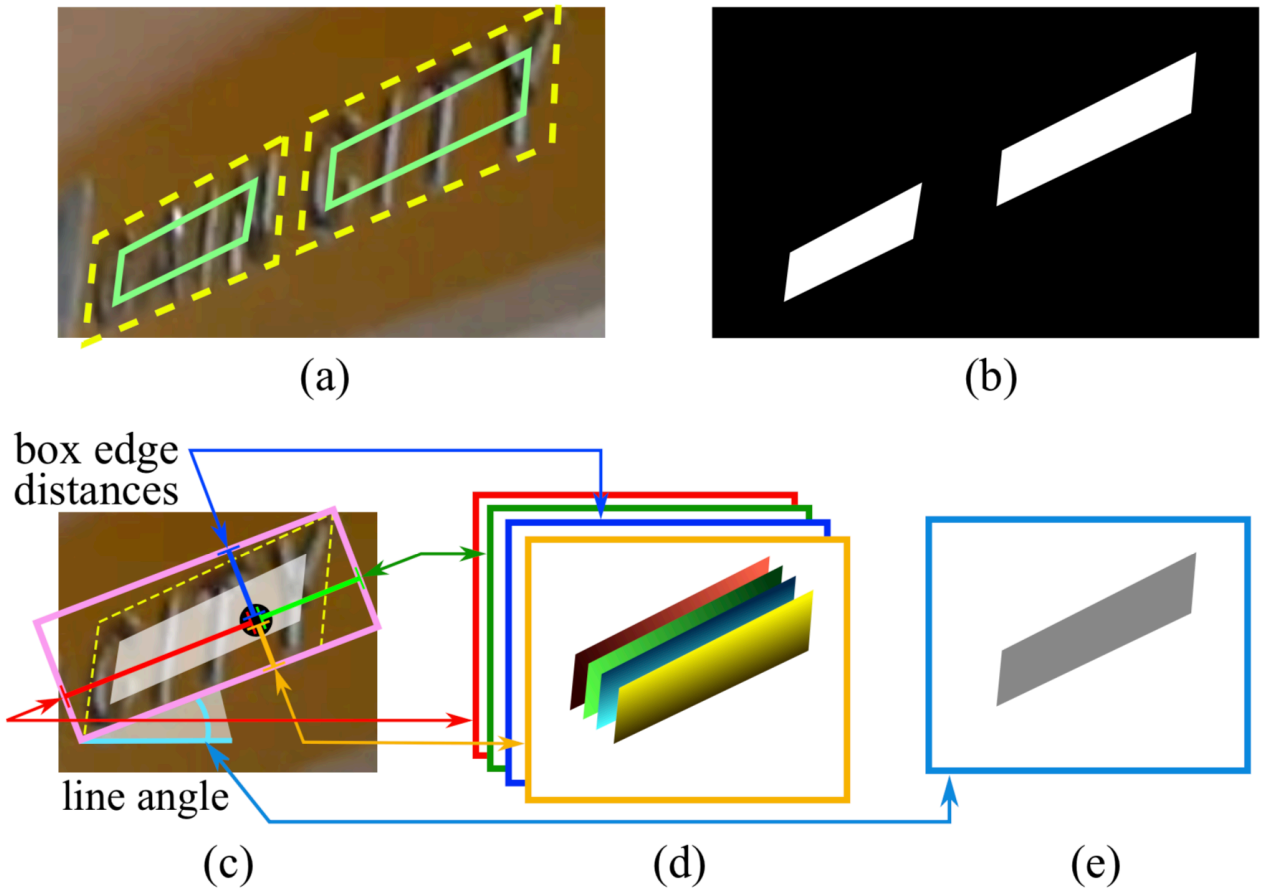
Label generation

- **Score map**
  - ① RBOX
  - ② QUAD
- **Geometry map**
  - ① RBOX
  - ② QUAD

Label generation

- **Score map**
  - ① RBOX
  - ② **QUAD**
- **Geometry map**
  - ① RBOX
  - ② **QUAD**

Label generation process



a. Text quadrangle (yellow dashed), shrunk quadrangle (green solid)

b. Text score map.

c. RBOX geometry map generation

d. 4-channels of distances of each pixel to rectangle boundaries

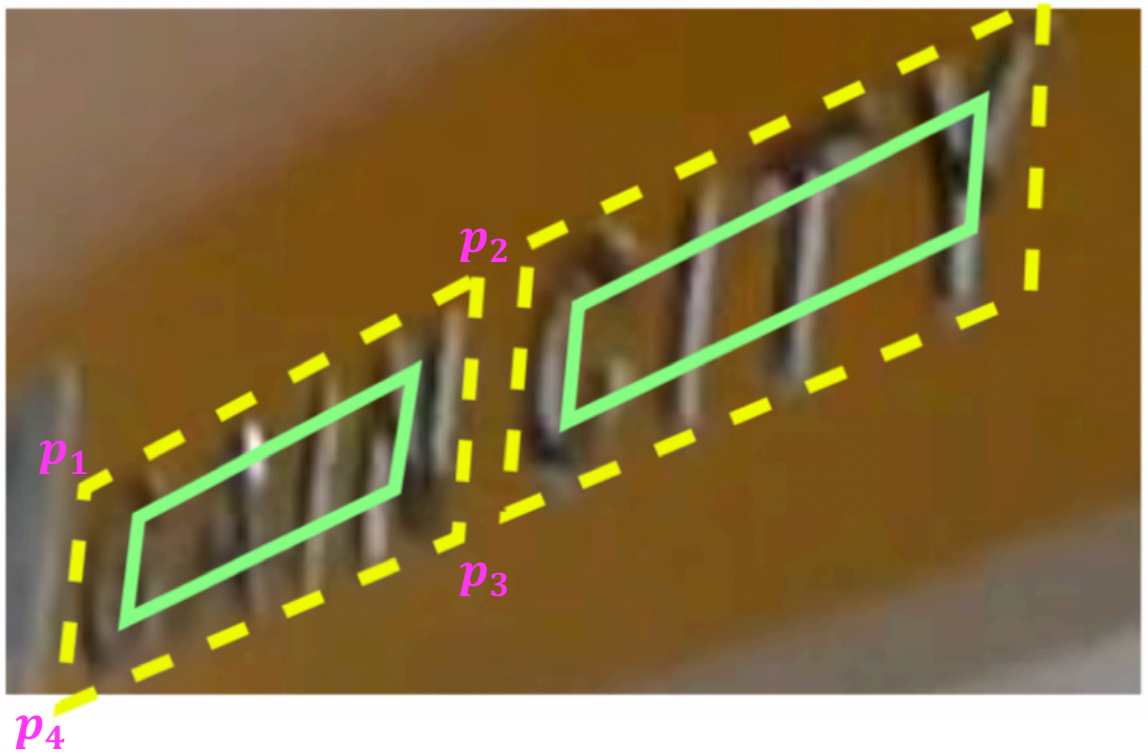
e. Rotation angle.

Score map generation for quadrangle

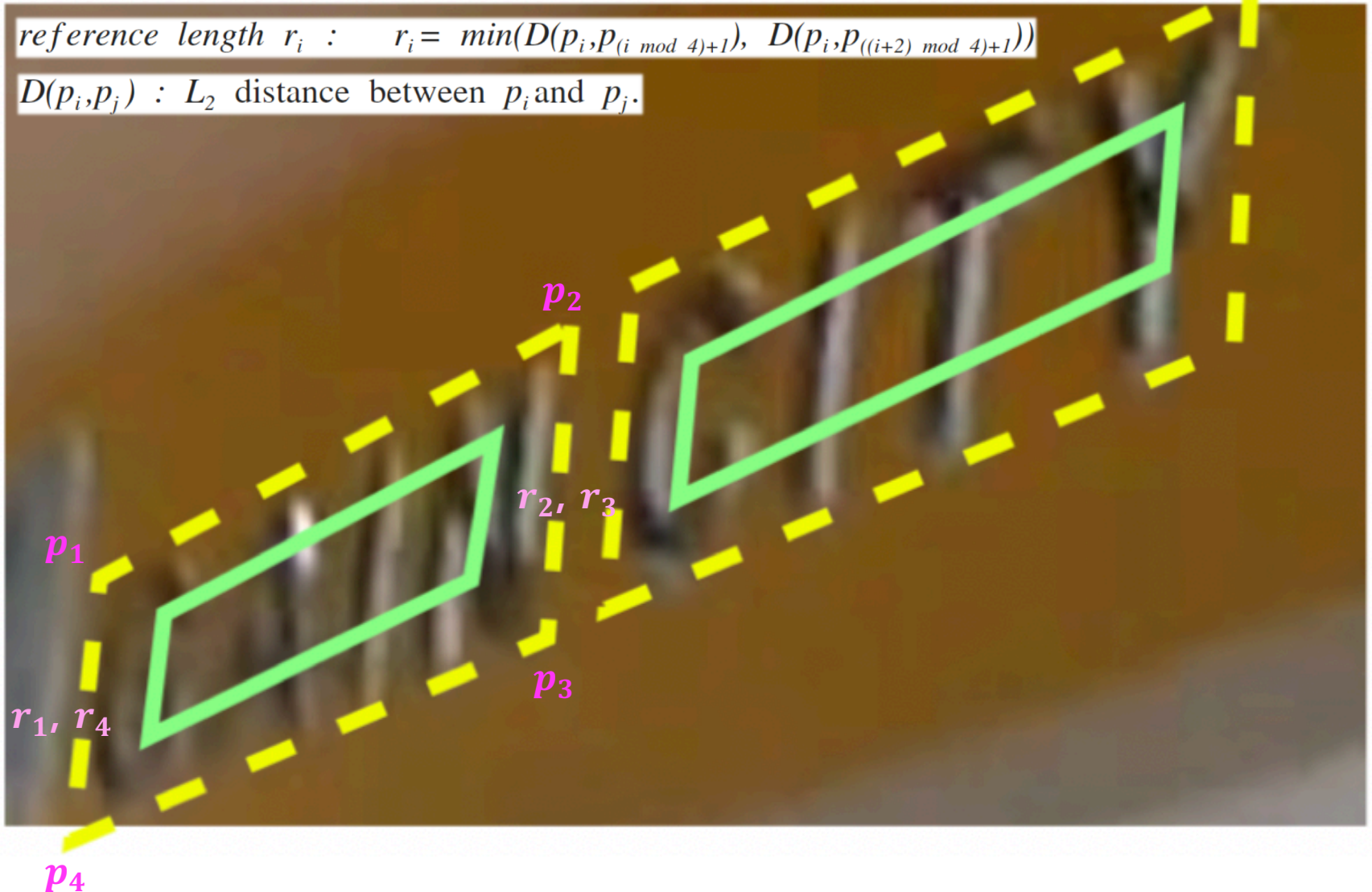
$$Q=\{p_i \mid i \in \{1,\ 2,\ 3,\ 4\}\},\ \ where\ \ p_i=\{x_i,y_i\}$$

$$reference\ \ length\ \ r_i\ :\quad r_i=\ min(D(p_i,p_{(i\ \ mod\ \ 4)+1}),\ \ D(p_i,p_{((i+2)\ \ mod\ \ 4)+1}))$$

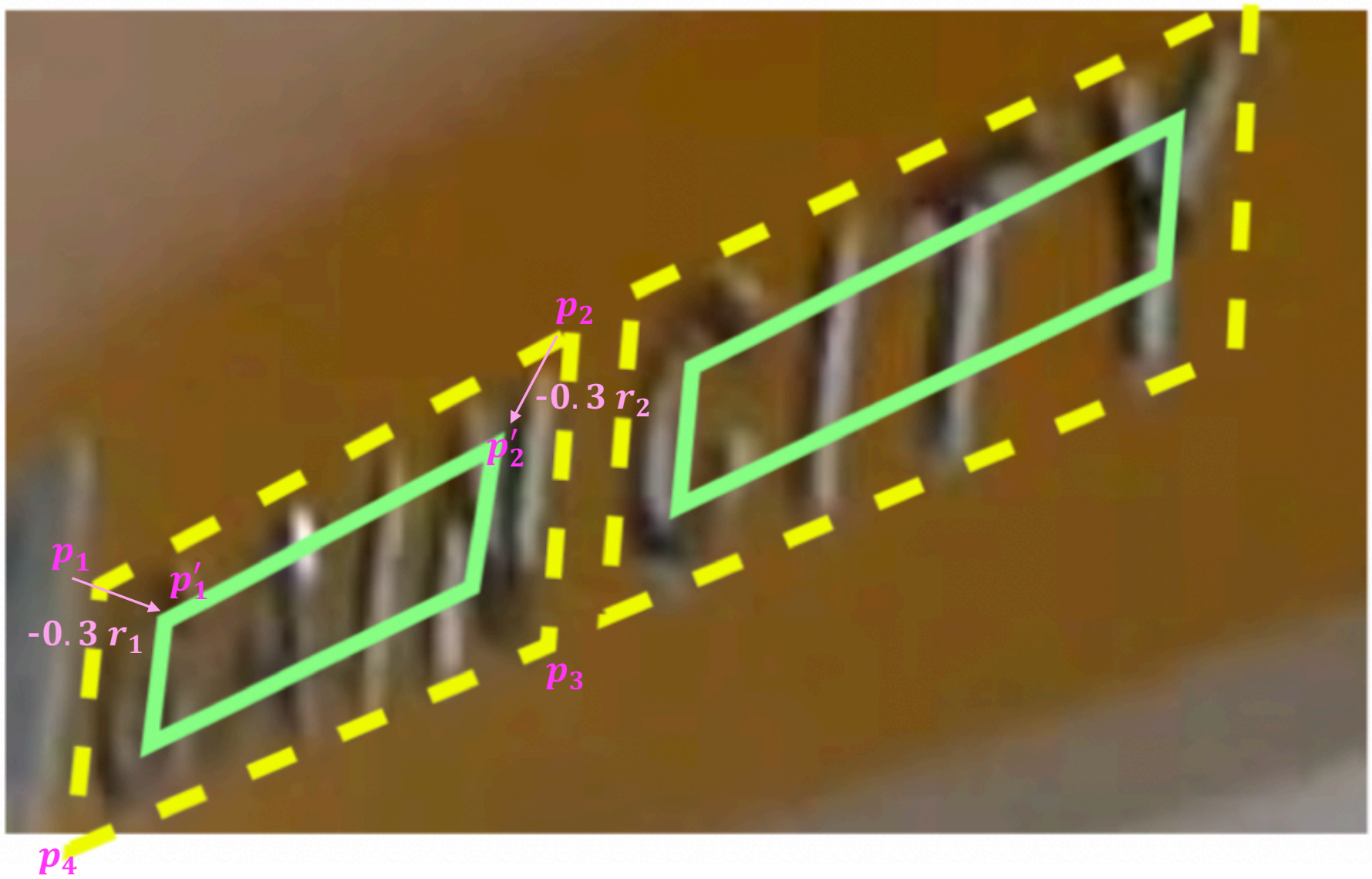
$$D(p_i,p_j)\ :\ L_2\ \text{distance between}\ p_i\ \text{and}\ p_j.$$



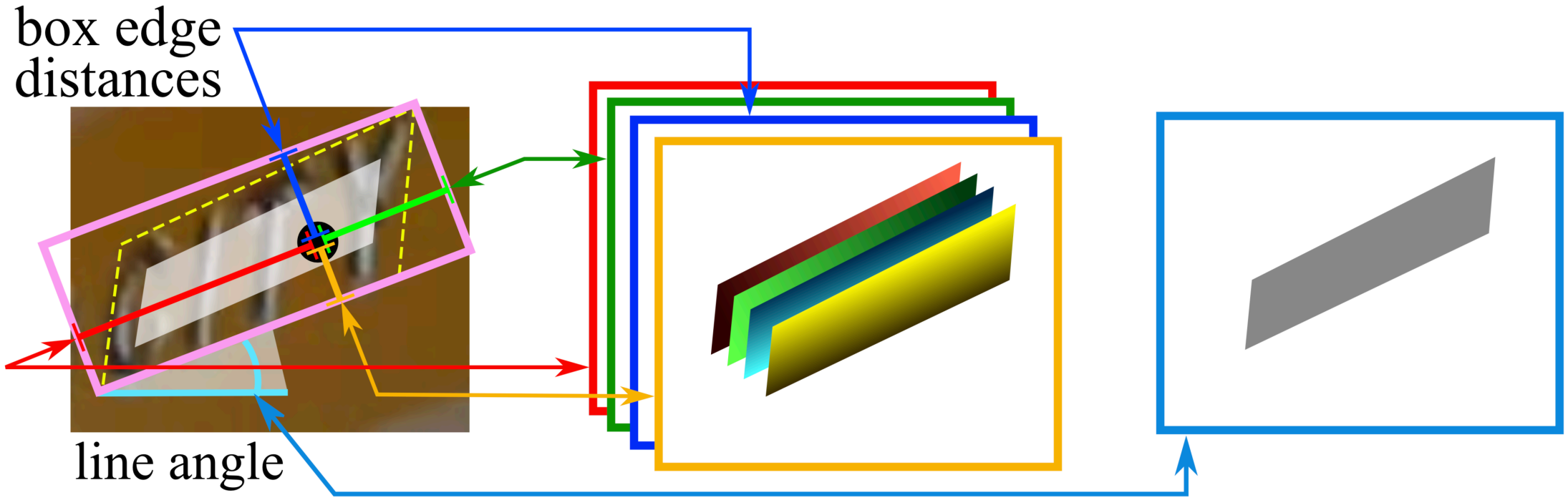
Score map generation for quadrangle



Score map generation for quadrangle



Geometry map generation



Loss functions

Loss functions

- **Score map**
- **Geometry map**
  - ① RBOX
  - ② QUAD

Two different loss functions

$$L = L_s + \lambda_g L_g$$

- Score map

$$L_{\mathrm{s}} = \text{balanced-xent}(\hat{\mathbf{Y}}, \mathbf{Y}^*)$$
$$= -\beta \mathbf{Y}^* \log \hat{\mathbf{Y}} - (1-\beta)(1-\mathbf{Y}^*) \log(1-\hat{\mathbf{Y}})$$

- Geometry map

$$L_{\mathrm{g}} = L_{\mathrm{AABB}} + \lambda_\theta L_\theta.$$

$$L_{\mathrm{AABB}} = -\log \mathrm{IoU}(\hat{\mathbf{R}}, \mathbf{R}^*) = -\log \frac{|\hat{\mathbf{R}} \cap \mathbf{R}^*|}{|\hat{\mathbf{R}} \cup \mathbf{R}^*|}$$

$$L_\theta(\hat{\theta}, \theta^*) = 1 - \cos(\hat{\theta} - \theta^*).$$

Loss functions – Score map

$$L = L_s + \lambda_g L_g$$

- Score map

prediction Ground Truth

$$L_s = \text{balanced-xent}(\hat{\mathbf{Y}}, \mathbf{Y}^*)$$
$$= -\beta \mathbf{Y}^* \log \hat{\mathbf{Y}} - (1-\beta)(1-\mathbf{Y}^*) \log(1-\hat{\mathbf{Y}})$$

Balancing !

$$L = L_s + \lambda_g L_g$$

- Score map

$$L_{\mathrm{s}} = \text{balanced-xent}(\hat{\mathbf{Y}}, \mathbf{Y}^*)$$
$$= -\beta \mathbf{Y}^* \log \hat{\mathbf{Y}} - (1-\beta)(1-\mathbf{Y}^*) \log(1-\hat{\mathbf{Y}})$$

balancing factor

$$\beta = 1 - \frac{\sum_{y^* \in \mathbf{Y}^*} y^*}{|\mathbf{Y}^*|}.$$

To tackle with imbalanced distribution.

- **Most state-of-the-art detection pipeline** : balanced sampling, hard negative mining
- **Ours** : balanced cross-entropy.

Loss functions – Geometry map

$$L = L_s + \lambda_g L_g$$

- Geometry map

$$L_{\mathrm{g}} = L_{\mathrm{AABB}} + \lambda_\theta L_\theta.$$

$$L_{\mathrm{AABB}} = -\log \mathrm{IoU}(\hat{\mathbf{R}}, \mathbf{R}^*) = -\log \frac{|\hat{\mathbf{R}} \cap \mathbf{R}^*|}{|\hat{\mathbf{R}} \cup \mathbf{R}^*|}$$

$$L_\theta(\hat{\theta}, \theta^*) = 1 - \cos(\hat{\theta} - \theta^*).$$

RBOX – IoU Loss

$$L = L_s + \lambda_g L_g$$

- Geometry map

$$L_{\mathrm{g}} = L_{\mathrm{AABB}} + \lambda_\theta L_\theta.$$

$$L_{\mathrm{AABB}} = -\log \mathrm{IoU}(\hat{\mathbf{R}}, \mathbf{R}^*) = -\log \frac{|\hat{\mathbf{R}} \cap \mathbf{R}^*|}{|\hat{\mathbf{R}} \cup \mathbf{R}^*|}$$

$$L_\theta(\hat{\theta}, \theta^*) = 1 - \cos(\hat{\theta} - \theta^*).$$

Why IoU ?

- **L1 or L2 loss for regression** : bias towards larger and longer text regions.
- **Ours** : IoU loss. (RBOX) , scale-normalized smoothed-L1 (QUAD)

QUAD – Scale normalized smoothed L1

$$L = L_s + \lambda_g L_g$$

- Geometry map

$$\mathrm{C}_{\mathbf{Q}} = \{x_1, y_1, x_2, y_2, \ldots, x_4, y_4\}$$

$$L_{\mathrm{g}} = L_{\mathrm{QUAD}}(\hat{\mathbf{Q}}, \mathbf{Q}^*) = \min_{\tilde{\mathbf{Q}} \in P_{\mathbf{Q}^*}} \sum_{\substack{c_i \in \mathrm{C}_{\mathbf{Q}}, \\ \tilde{c}_i \in \mathrm{C}_{\tilde{\mathbf{Q}}}}} \frac{\mathrm{smoothed}_{L1}(c_i - \tilde{c}_i)}{8 \times N_{\mathbf{Q}^*}}$$

$$N_{\mathbf{Q}^*} = \min_{i=1}^{4} D(p_i, p_{(i \bmod 4)+1}),$$

Post-processing
(thresholding & NMS)

Locality – Aware NMS

Naïve NMS vs **Locality-Aware NMS**

- **Naïve NMS** : $O(n^2)$
- **Locality-Aware NMS** : $best: O(n), worst\ O(n^2)$

Assumption: geometries from nearby pixels tend to be highly correlated.

→ merge the geometries row by row

Algorithm 1 Locality-Aware NMS

```
function NMSLocality(geometries)
    S ← ∅,  p ← ∅
    for g ∈ geometries in row first order do
        if p ≠ ∅ ∧ ShouldMerge(g, p) then
            p ← WeightedMerge(g, p)
        else
            if p ≠ ∅ then
                S ← S ∪ {p}
            end if
            p ← g
        end if
    end for
    if p ≠ ∅ then
        S ← S ∪ {p}
    end if
    return StandardNMS(S)
end function
```

Experiments

Result – ICDAR 2015

Algorithm	Recall	Precision	F-score
Ours + PVANET2x RBOX MS*	**0.7833**	0.8327	**0.8072**
Ours + PVANET2x RBOX	0.7347	**0.8357**	**0.7820**
Ours + PVANET2x QUAD	0.7419	0.8018	0.7707
Ours + VGG16 RBOX	0.7275	0.8046	0.7641
Ours + PVANET RBOX	0.7135	0.8063	0.7571
Ours + PVANET QUAD	0.6856	0.8119	0.7434
Ours + VGG16 QUAD	0.6895	0.7987	0.7401
Yao *et al.* [41]	0.5869	0.7226	0.6477
Tian *et al.* [34]	0.5156	0.7422	0.6085
Zhang *et al.* [48]	0.4309	0.7081	0.5358
StradVision2 [15]	0.3674	0.7746	0.4984
StradVision1 [15]	0.4627	0.5339	0.4957
NJU [15]	0.3625	0.7044	0.4787
AJOU [20]	0.4694	0.4726	0.4710
Deep2Text-MO [45, 44]	0.3211	0.4959	0.3898
CNN MSER [15]	0.3442	0.3471	0.3457

Approach	Res.	Device	T_1/T_2 (ms)	FPS
Ours + PVANET	720p	Titan X	58.1 / 1.5	16.8
Ours + PVANET2x	720p	Titan X	73.8 / 1.7	13.2
Ours + VGG16	720p	Titan X	150.9 / 2.4	6.52
Yao *et al.* [41]	480p	K40m	420 / 200	1.61
Tian *et al.* [34]	ss-600*	GPU	130 / 10	7.14
Zhang *et al.* [48]*	MS*	Titan X	2100 / N/A	0.476

Thank you.